Schedulability Analysis for Desktop Multimedia Applications: Simple Ways to Handle General-Purpose Operating Systems and Open Environments

Erik Cota-Robles (Erik_Cota-Robles@ccm.jf.intel.com)
James P. Held (Jim_P_Held@ccm.jf.intel.com)
Thomas J. Barnes (Thomas_J_Barnes@ccm.jf.intel.com)
Intel Architecture Labs
Intel Corporation
M/S JF2-55
2111 N.E. 25th Ave.
Hillsboro, OR 97124-5961

Abstract

Unlike traditional real-time applications, multimedia applications are commonly deployed on top of general-purpose operating systems in open environments. Because Rate Monotonic Analysis (RMA) assumes that the total system workload is known in advance and uses worst case values for operating system overhead (e.g., context switch time), it has not been usable for personal computer and workstation based multimedia applications. We extend RMA to model systems where these OS services have highly non-deterministic timing behavior and where the total system workload is not specified in advance. We validate our techniques by modeling a video telephony application based on Intel's ProShare® technology.

1. Introduction

In recent years, real-time multimedia and conferencing applications have become increasingly common on personal computers and desktop workstations. With the advent of more powerful microprocessors, it has become feasible to perform significant time-critical processing in software on the host; this processing was previously relegated to specialized hardware. This trend is expected to continue with recent advances in microprocessor technology, such as the introduction of the MMX™ instruction set for the Pentium® and Pentium® II processors.

In a typical videoconferencing application, such host-based real-time processing can include both video and audio codecs as well as acoustic echo cancellation. Since for this class of application some video jitter is tolerable to the end user, we consider video requirements to be soft real-time. In contrast, since even small amounts of audio packet loss or delay rapidly render speech incomprehensible, we consider audio requirements to be hard real-time. As another example of a hard real-time requirement, future personal computers may include a Digital Simultaneous Voice and Data (DSVD) modem with a data pump implemented wholly in software. Future personal computers applications with host-based real-time processing may include speech recognition and home entertainment.

Traditionally, real-time applications have been deployed on dedicated systems running special-purpose real-time operating systems which utilize priority-based scheduling and feature deterministic response times for system services such as task context switch time. In contrast, desktop workstation operating systems have generally utilized time sharing scheduling policies such as round robin and FIFO, although newer systems such as Microsoft Windows[*] NT* do make explicit provision for real-time priority classes. On such systems, the response times for system services can be highly non-deterministic. As a result, problems arise when attempting to guarantee that real-time applications will meet their timing requirements, because the worst case timings for system services are so large that worst case timing analyses are unduly pessimistic.

---

[*] Other product and corporate names may be trademarks of other companies and are used only for explanation and to the owners' benefit, without intent to infringe.

In this paper, we will only be concerned with preemptive fixed-priority scheduling, which is widely available in commercial real-time operating systems, and increasingly in desktop workstation and personal computer operating systems. Fixed-priority scheduling theory has undergone dramatic changes during the past ten years. The seminal work on real-time scheduling theory by Liu and Layland [11] was limited in applicability as it applied only to periodic task sets that do not share resources and assumed that operating system overhead was negligible. They proved that assigning task priorities so that short period tasks have priority over tasks with longer periods (i.e., in rate monotonic order) is optimal, and they derived a sufficient but not necessary condition for determining whether a set of such tasks will meet their deadlines under all conditions. We refer to tasks and applications as being schedulable if they will meet their deadlines under all conditions.

Subsequent work has removed or relaxed many of the assumptions made in [11]. In [18] the theory is extended to handle aperiodic tasks which have a minimum interarrival time by creating a "Sporadic Server" task which is analyzable because it is periodic. The ability to share resources was added via the techniques of "priority inheritance" [5][17], whereby the priority of a task is temporarily elevated when it holds a resource which a higher priority task requires. This serves to bound the time during which the higher priority task is blocked from executing by a lower priority task, which is distinct from (and worse than) preemption, where a task is kept from executing by higher priority task. Methods of accounting for operating system overhead are discussed in [6] and [9] using worst case timings for system services such as task context switch time. In this work we develop extensions to RMA that enable us to model real-time applications on general-purpose operating systems.

The remainder of this paper is organized as follows. Section 2 provides necessary background about Rate Monotonic Analysis (RMA). In section 3, we discuss issues of performing timing analyses on real-time systems that must execute on general-purpose operating systems in open environments. Section 4 presents our extensions to RMA, which enable us to model real-time applications on desktop systems. In section 5, we describe how we applied these techniques using the University of Illinois' Prototyping Environment for Real-Time Systems (PERTS) [12] to automate RMA calculations. Section 6 describes a model of a video telephony application based on Intel's ProShare® technology and compares predicted with actual application behavior. We conclude in section 7.

2. Rate Monotonic Analysis

Rate Monotonic Analysis begins with the observation that for any schedulable unit of computation, or task, the sum of the following times must be less than the interval until the task's deadline:

- The task execution or computation time (i.e., the time for the task itself to execute all of its instructions).
- The time that the task is preempted from executing by higher priority tasks.
- The time that the task is blocked from executing by lower priority tasks.
- The time that the operating system executes on behalf of the task or other tasks (context switches, interrupt servicing, etc.)

RMA is applied in practice by iteratively applying schedulability tests derived from rate monotonic scheduling theory to a set of periodic tasks [10]. Sporadic tasks, which are not periodic but do have minimum interarrival times, are modeled by periodic server tasks which have a fixed execution budget per period [18]. At each iteration, the tasks that are still not schedulable are identified, and candidates for reduction (e.g., tasks which are blocked for long periods of time) are chosen. After any possible reductions are achieved (e.g., by reducing resource sharing) the process is repeated until all tasks are schedulable or no further reductions are feasible.

A number of tests are available to verify schedulability of a task. The most important of these is the Utilization Bounds test [10], which is based on the following theorem due to Liu and Layland [11]:

Thm. 1: Given n independent tasks, let task t_i have period T_i and execution time C_i. Then if the total utilization

$$U_{total} = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \dots + \frac{C_n}{T_n} \le U(n)$$

of all tasks is no more than $U(n) = n(2^{1/n} - 1)$ then all n tasks are schedulable, excluding external events (e.g., operating system actions), provided that the task priorities are assigned in rate monotonic order.

Theorem 1 assumes that all of the tasks are independent (i.e., there is no blocking due to synchronization). A corollary, due to Sha, Rajkumar and Lehoczky [17], extends the test to cover tasks that share data or otherwise must synchronize execution:

Cor. 1: Given n tasks, let task t_i have period T_i, execution time C_i and worst case blocking time B_i. Then if the total utilization, including worst case blocking time,

$$U_{total} = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \Lambda + \frac{C_n}{T_n} + \max_{i=1}^{n} \frac{B_i}{T_i} \le U(n)$$

of all tasks is no more than $U(n) = n(2^{1/n} - 1)$ then all n tasks are schedulable, again excluding operating system actions, provided that the task priorities are assigned in rate monotonic order.

In order for the worst case blocking time $\max_{i=1}^{n}(B_i / T_i)$ to be bounded, the operating system must protect against unbounded priority inversion by using one of the priority inheritance protocols such as the Priority Ceiling Protocol [17], the Highest Locker Protocol [10] or the Stack-Based Protocol [2]. Although such protocols are commonly available with real-time operating systems, they are not widely available on personal computer and desktop workstation operating systems.

Application of the Utilization Bounds Test is relatively straightforward. For the case where the task periods are all harmonic the bound is in fact 1.0 [11]. In general, $U(n) = n(2^{1/n} - 1) \ge \ln(2) = 0.693$. The test is sufficient but not necessary to guarantee schedulability and has three possible outcomes:

- Schedulable if. $0 \le U_{total} \le U(n)$
- Not schedulable if $U_{total} > 1.0$.
- Unknown if $U(n) < U_{total} \le 1.0$.

The test can be modified to include task context switch overhead by including the worst case task context switch time, C_s, in the execution time of each task. The following corollary is due to Klein and Ralya [9]:

Cor. 2: Given n tasks, let task t_i have period T_i, execution time C_i and worst case blocking time B_i. Let the operating system have worst case task context switch time C_s, and for each task t_i let $C'_i = C_i + 2C_s$ be the execution time including (worst case) context switch time. Then if the total utilization, including worst case context switch time,

$$U_{total} = \frac{C'_1}{T_1} + \frac{C'_2}{T_2} + \Lambda + \frac{C'_n}{T_n} + \max_{i=1}^{n} \frac{B_i}{T_i} \le U(n)$$

of all tasks is no more than $U(n) = n(2^{1/n} - 1)$ then all n tasks are schedulable, including the cost of context switches but excluding all external events, provided that the task priorities are assigned in rate monotonic order.

It is necessary to charge each task for two context switches, one to begin execution and one to complete execution, since in the worst case each task preempts a task of lower priority. The analysis still ignores other operating system overheads such as time to service interrupts, but if the necessary data are available these can be modeled as sporadic tasks with appropriate minimum interarrival times and maximum execution times.

Unfortunately the Utilization Bounds test is often overly pessimistic. For tasks that execute at a single priority, the Response Time test [10] is an exact schedulability test. For tasks that execute at varying priorities (e.g., because one of the priority inheritance protocols is being used to bound blocking), the techniques of [6] can be used. An abbreviated algorithm for the Response Time test is as follows:

For each task t_i with deadline D_i do:

1) Initialize $a_0 = B_i + \sum_{j=1}^{i} C_j$, $n = 0$ and $k = 1$.
2) $a_{n+1} = B_i + kC_i + \sum_{j=1}^{i-1}((a_n + T_i - 1) / T_j) \cdot C_j$.
3) If $a_{n+1} > (k-1)T_i + D_i$ then terminate (failure).
4) Else if $a_{n+1} \ne a_n$ increment n and goto 2.
5) $E_{i,k} = a_n - (k-1)T_i$.
6) If $E_{i,k} > T_i$ but $E_{i,k} < D_i$, set $a_{n+1} = a_n + C_i$, increment n and k and goto 2.
7) If $E_{i,k} \le T_i$ and $E_{i,k} \le D_i$ for all k then task t_i is schedulable.

In practice, analyses of this complexity are best left to automated tools, a number of which have been developed in recent years [7][12][14][19]. Such tools can also incorporate worst case task context switch times in the analysis by using the techniques of [6] and modeling each task as a supertask with varying priority.

3. Timing Analysis of Real-Time Applications on Desktop Operating Systems

Previously the rate monotonic analysis methodology has not been applicable to real-time applications that must execute on personal computers and desktop workstations. There are two main reasons for this:

- General-purpose operating systems have virtual memory and other features that cause worst case times for system services (e.g., context switching) to be orders of magnitude worse than the average case.
- The execution time environment for the application, both in terms of processor load from other applications and in terms of hardware resources (e.g., cache size, miss penalty), is not known in advance.

Many factors determine the real-time response of an operating environment. On the part of the operating system, these include the longest critical region during which the kernel blocks task activation during a system call, the maximum time that interrupts are disabled in the kernel, and the performance of synchronization services (e.g., semaphores) and interprocess communication (IPC) [15]. On the part of the underlying hardware, these include both the time to save and restore register and memory contexts as well as the type and speed of the processor, cache, and memory. Finally, the overall application load at run time profoundly influences cache miss and memory page fault rates.

Previous efforts to quantify performance of personal computer and desktop workstation operating systems have focused on average case values and measurements conducted on an otherwise unloaded system [4][16]. In particular, Ousterhout [16] evaluates operating system performance using a collection of microbenchmarks, including time to enter/exit the kernel, process context switch time, and several file I/O benchmarks. While it is no reflection on the validity of Ousterhout's study for the purposes for which it was intended, all results are reported in terms of average acess time which is not the primary metric of interest for real-time applications. For real-time applications the metrics of interest are the worst case time and the full probability distribution. In contrast, Endo, et. al. [4] develop microbenchmarks based on simple interactive events such as key stroke and mouse click, as well as task-oriented benchmarks designed to model specific user actions when using popular applications such as Microsoft Word*. Although detailed distributions are reported, they are for otherwise unloaded systems. In addition, none of the microbenchmarks directly address response to interrupts, which is of prime importance to periodic real-time applications.

Efforts to characterize the behavior of real-time systems, both software and hardware, have focused largely on worst case behavior and assumed that the overall system load is known in advance. Klein and Ralya [9] develop a schedulability analysis methodology for a cyclic input/processing/output model of computation in conjunction with both synchronous and asynchronous I/O paradigms. Katcher, et. al. [8] decompose operating system overhead into the four categories of preemption, (task) exit, nonpreemption and timer interrupts, adopting the following notation: $C_{preempt}$ is the time to preempt a task; C_{exit}, the time to resume execution of a previously preempted task; $C_{nonpreempt}$, blocking time due to interrupt handling which does not result in task preemption (i.e., the task is added to the ready queue); and C_{timer}, the time to service a timer interrupt. For the event driven scheduling paradigm $C_{preempt}$ includes the time to service an interrupt, save task context, and schedule and load a new task, while C_{exit} includes time to trap to the scheduler and load the next task on the ready queue. While this model is comprehensive and adequate for real-time operating systems, it is overly pessimistic for personal computer and desktop workstation operating systems which frequently have worst case times for system services such as context switching that are orders of magnitude longer than average case times. We sought a small set of microbenchmarks that would encapsulate the effects of the operating environment but could be manageably incorporated into an RMA analysis. Since our goal was for the benchmarks to be applicable to a variety of real-time applications we avoided task-oriented benchmarks in favor of general microbenchmarks.

Context switch time, our first microbenchmark, is very similar to Ousterhout's cswitch benchmark [16], but uses semaphores for IPC. A pair of real-time tasks share a semaphore. Periodically one task reads the Pentium processor cycle count register and sets the semaphore. The other task, which has been pending on the semaphore, reads the cycle count register and calculates the elapsed time. This benchmark incorporates the performance of semaphores as well as the effects of cache misses as task context is restored by the operating system. For user mode tasks, the cost to enter and exit the kernel is also encompassed. In the terminology of Katcher, et. al. [8] this benchmark measures C_{exit}, the time to trap to the scheduler and load the next task on the ready queue, as well as the time to schedule a new task since the task in question was previously pending on the semaphore.

Interrupt latency and task latency, our other two microbenchmarks, quantify system responsiveness to interrupts, which is very important for real-time systems. Interrupt latency is the worst case latency between the occurrence and execution of the first instruction of the interrupt handler, while task latency is the worst case task dispatch latency for a task waiting on an interrupt (i.e., the delay between the handler and first instruction of the task). Together these benchmarks measure Katcher, et. al.'s $C_{preempt}$, the time to service an interrupt, save task context, and schedule and load a new task. In the case of task latency, the task was again a real-time task that waited on an interrupt and then read the cycle count register and calculated the elapsed time using a reading taken by the interrupt handler. Interrupt latency incorporates the maximum time interrupts are disabled, while task latency incorporates the performance of semaphores as well as the cost of exiting the kernel and some effects of cache misses and page faults as context is restored by the operating system. In practice we found that on our systems the worst case interrupt latency was neglible in comparison with the worst case task latency, but it must be borne in mind that on an open system an ill-behaved device driver can disable interrupts for arbitrary periods.

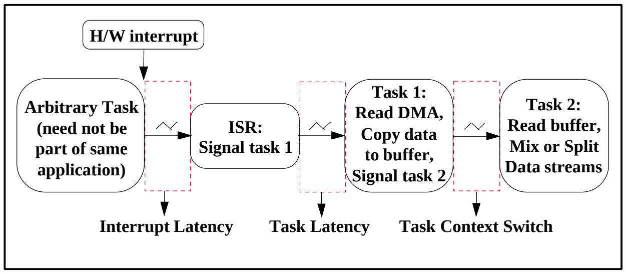


Figure 1: Interrupt Latency, Task Latency and Task Context Switch Time

For a typical multimedia application the bulk of the real-time processing cannot be done in an interrupt service routine, and since the data is typically handed off from task to task, all three measures are important to determining whether a multimedia application, particularly a streaming one, will be able to meet its deadlines and thus deliver the required quality of service. This is depicted in Figure 1.

Since the execution time environment for the application in terms of processor load from other applications is unknown, we characterized the effective context switch time and interrupt latency under load on several varieties of Microsoft's Windows operating systems. The measurements were performed using kernel device drivers for Windows NT and ring 0 VxDs for Windows 95 together with a Win32 (user mode/ring 3) control application. Load was imposed by running BAPCo System's Sysmark32 benchmark [3], a standard PC benchmarking tool which runs a mix of desktop workstation applications. Context switch time and interrupt and task latency were measured every 25 ms. while running the Sysmark benchmark.

Measured Windows	Windows NT		Windows 95		
Service Times	User	Kernel	Ring 3	Ring 0	RT Kernel
Median Context Switch Time	48us	29us	110us	54us	26us
Worst Case Context Switch Time	9.9ms	9.9ms	218ms	4.0ms	3.5ms
Median Task Latency	110us	30us	98us	59us	43us
Worst Case Task Latency	9.7ms	9.6ms	169ms	109ms	3.5ms
Median Interrupt Latency		<10us		<<100us	
Worst Case Interrupt Latency		50us		500us	

Figure 2: Measured Windows Service Times

We ran our benchmarks on a personal computer with an Aladdin motherboard configured with a 100 Mhz Pentium processor, a Triton chip set with 256K L2 cache, 16 MB of fast page DRAM and an ATI Mach 64 4M running at 800x600x256. We found that for Windows NT worst case times for both thread context switches and thread latency were two orders of magnitude larger than the average case, ranging up to 9.9 ms., excluding outliers, while for Windows 95 worst case times were three orders of magnitude larger. A real-time kernel which runs as a Windows 95 VxD improved performance considerably. Figure 2 gives the figures for task context switch time and task latency for both user and kernel modes (Windows NT) and ring 3, ring 0 and the real-time kernel for Windows 95, as well as for interrupt latency for Windows NT and Windows 95. For Windows 95 the worst case interrupt latency has been adjusted to exclude extremely long interrupt latencies observed when video drivers switched between Windowed GDI and DOS full-screen modes, an event intrinsically incompatible with uninterrupted streaming multimedia.

A typical streaming multimedia application will have task deadlines (e.g., for audio) much less than 100 ms., so the immediate conclusion is that no application requiring high quality uninterrupted real-time audio will be guananteed schedulable using ring 3 Win32 threads on Windows 95. However, even for Windows NT, standard worst case analysis indicates that a system whose average utilization is very light can easily fail to be schedulable. For both operating systems an analysis using average case values for system services fails to predict transient overloads which will cause missed deadlines, loss of service and, in the case of streaming multimedia applications, video jitter, audio drop-outs or worse. Because the worst case interrupt latencies are orders of magnitude smaller than the worst case task latencies, we simplified our analyses by subsuming interrupt latency into task latency.

4. Extending RMA to Desktop Workstations

In response to these difficulties we devised a method of performing "reasonable" worst-case analysis, which gives meaningful answers in situations where true worst-case analysis fails to give usable results. We hypothesized that the extremely long observed context switch times and task latencies that occur under load on Windows 95 and Windows NT represent exceptional conditions which are highly unlikely to occur back to back, or even within 100 or 200 ms. of each other. Indeed, because of the low frequency of the very worst events, a hypothesis of statistical independence would result in the same conclusion. While it seems reasonable to assume that during any task's active period (i.e., between its release time and its deadline) at most one worst case context switch or task latency will occur, one must still consider whether a task can still meet its deadline if a single one does occur. In the lemma and corollary below we use context switch time as shorthand for the larger of the (average/worst case) task switch time and task latency.

Lemma 1: Given n tasks, let task t_i have period T_i, execution time C_i and worst case blocking time B_i. Let the operating system have worst case task context switch time C_s^W and average case task context switch time C_s^A, and for each task t_i let $C'_i = C_i + 2C_s^A$ be the execution time including (average case) context switch time. Then if for for each task t_k, $1 \le k \le n$, the total utilization including one worst case context switch time

$$U_k = \frac{C'_1}{T_1} + \frac{C'_2}{T_2} + \cdots + \frac{C'_k}{T_k} + \max\left(\max_{i=1}^{k-1} \frac{B_i}{T_i}, \frac{(B_k + C_s^W - C_s^A)}{T_k} \right) \le U(k)$$

is no more than $U(k) = k(2^{1/k} - 1)$ then all n tasks are schedulable, provided that no more than a single worst case context switch or task latency occurs during any task's active period and that all other context switches are of no more than average cost, and also, of course, that the task priorities are assigned in rate monotonic order.

Proof: Since we have assumed that all other context switches are of average cost, we can regard C_s^A as the effective context switch time. If we then check each task t_k, $1 \le k \le n$, in turn we can regard $C_s^W - C_s^A$ as additional blocking time for t_k and the result follows from corollary 2.

Current tools for performing rate monotonic analysis use only worst case context switch times, which makes it hard to apply lemma 1 directly. The following corollary amortizes the cost of the worst case context switch over all of the average case ones, giving a pseudo worst case context switch time which can be entered into tools designed for real-time operating systems:

Cor 3: Given n tasks, let task t_i have period T_i, execution time C_i and worst case blocking time B_i. Let the operating system have worst case task context switch time C_s^W and average case task context switch time C_s^A. For each task t_k let K be the minimum number of context switches which must occur on behalf of t_k and higher priority tasks during t_k's period. For each task t_i, $1 \le i \le k$, let $C''_i = C_i + 2 \cdot \frac{C_s^W + (K-1)C_s^A}{K}$ be the execution time including amortized worst case context switch time. Then if for for each task t_k, $1 \le k \le n$, the total utilization including pseudo worst case context switch time

$$U_k = \frac{C''_1}{T_1} + \frac{C''_2}{T_2} + \cdots + \frac{C''_k}{T_k} + \max_{i=1}^{k} \frac{B_i}{T_i} \le U(k)$$

is no more than $U(k) = k(2^{1/k} - 1)$ then all n tasks are schedulable, provided that no more than a single worst case context switch or task latency occurs during any task's active period and that all other context switches are of no more than average cost.

Proof: For $k = 1$ this is lemma 1. Since $C_s^A \le C_s^W$, $C_s^A \le \frac{C_s^W + (K-1)C_s^A}{K}$, and because $T_{i-1} \le T_i$, $1 \le i \le n$, if we set $C_s^Z = \frac{C_s^W + (K-1)C_s^A}{K}$ it follows that

$$\frac{C_{i-1} + 2C_s^Z}{T_{i-1}} + \frac{C_i + 2C_s^Z}{T_i} \ge \frac{C_{i-1} + 2C_s^A}{T_{i-1}} + \frac{C_i + 4C_s^Z - 2C_s^A}{T_i}$$

Then, letting $C'_i = C_i + 2C_s^A$ as in lemma 1, since $\frac{C_s^W - C_s^A}{T_k} + \max_{i=1}^{k} \frac{B_i}{T_i} \ge \max\left(\max_{i=1}^{k-1} \frac{B_i}{T_i}, \frac{B_k + C_s^W - C_s^A}{T_k} \right)$, it follows by induction that

$$\frac{C''_1}{T_1} + \frac{C''_2}{T_2} + \cdots + \frac{C''_k}{T_k} + \max_{i=1}^{k} \frac{B_i}{T_i} \ge$$

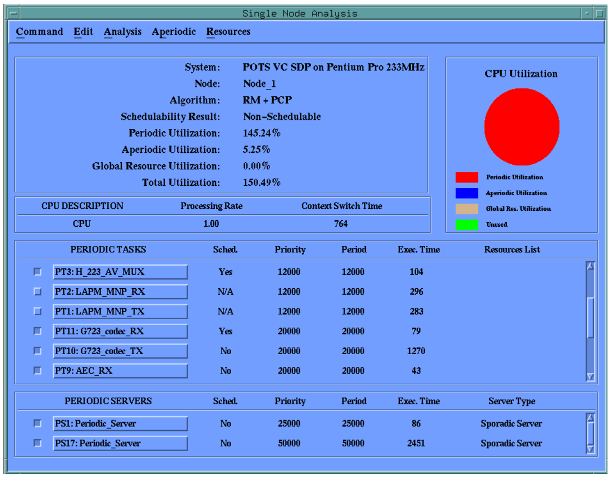
$$\frac{C'_1}{T_1} + \cdots + \frac{C'_k}{T_k} + \max\left(\max_{i=1}^{k-1} \frac{B_i}{T_i}, \frac{(B_k + C_s^W - C_s^A)}{T_k} \right)$$

and the result follows from lemma 1.

5. Modeling Desktop Multimedia Applications

Using the data on the observed distribution under load of the task context switch time and task latency, we first perform a standard rate monotonic analysis using average case operating system service times in order to determine the overall system utilization. As shown in Figure 2, on Windows NT average (median) case times in user mode for context switch time and task latency are 48 us. and 110 us., respectively, so we simply used the latter. To perform the analysis we used the University of Illinois' Prototyping Environment for Real-Time Systems (PERTS) [12]. Other RMA tools include CAISARTS [7], PerfoRMAx [14], SEW [19] and STRESS [1].

To perform the Pseudo Worst Case Analysis we amortized the cost of a single instance of the worst case observed context switch time over the minimum number of task switches that must occur during the period of each task (see corollary 3, above). In performing the amortization we sum the worst case context switch time and $n-1$ average (median) case context switch times, where n is the minimum number of task switches that must occur during a given period (e.g., 12 ms. for a software data pump) with only the task and any higher priority tasks running. We then divide this by n before applying rate monotonic analysis because when applying the Utilization Bounds Test and allowing for context switch overhead in RMA the worst case context switch time is doubled. The resulting pseudo worst case task context switch time is simply entered into an automated RMA tool as the worst case context switch time. This procedure must be repeated for each task.



System: POTS VC SDP on Pentium Pro 233MHz
Node: Node_1
Algorithm: RM + PCP
Schedulability Result: Non–Schedulable
Periodic Utilization: 145.24%
Aperiodic Utilization: 5.25%
Global Resource Utilization: 0.00%
Total Utilization: 150.49%

CPU DESCRIPTION	Processing Rate	Context Switch Time
CPU	1.00	764

PERIODIC TASKS	Sched.	Priority	Period	Exec. Time	Resources List
☐ PT3: H_223_AV_MUX	Yes	12000	12000	104	
☐ PT2: LAPM_MNP_RX	N/A	12000	12000	296	
☐ PT1: LAPM_MNP_TX	N/A	12000	12000	283	
☐ PT11: G723_codec_RX	Yes	20000	20000	79	
☐ PT10: G723_codec_TX	No	20000	20000	1270	
☐ PT9: AEC_RX	No	20000	20000	43	

PERIODIC SERVERS	Sched.	Priority	Period	Exec. Time	Server Type
☐ PS1: Periodic_Server	No	25000	25000	86	Sporadic Server
☐ PS17: Periodic_Server	No	50000	50000	2451	Sporadic Server

Figure 3: PERTS Schedulability Analyzer, Showing Test for Audio Drop-outs in Video Telephony

As an example, for the video teleconferencing application described in section 6, we tested the audio tasks for schedulability as follows. There are 5 modem tasks (4 in kernel mode) and 2 audio tasks. The modem tasks have a period of 12 ms. and since $\lfloor 20/12 \rfloor = 1$, we have the following:

$$\frac{9900+2\cdot(2.5\cdot 110+4\cdot 30)}{(5+2)\cdot 2}=\frac{10690}{14}=764\ us$$

Note that we have added in 13 additional task switches, since $7\cdot 2=14$ is the worst case number. Figure 3 shows this value (764 us.) being used to test the Video teleconferencing application on a Pentium Pro processor for audio drop-outs.

Because the worst case task context switches and task latencies are primarily caused by phenomena external to the processor we investigated the scaling of these worst case events on Pentium processor based systems at a range of speeds. Investigation revealed that the worst case time scales little if at all with Pentium processor MHz although the average case times do scale. We therefore decided to assume no reduction in worst case times for our analyses.

6. Modeling a Video Teleconferencing Application with Software Data Pump

We tested our methodology by creating a model of a hypothetical video teleconferencing application which could run on Windows NT on Pentium and Pentium Pro processor based systems with MMX Technology and use a modem implemented wholly in software (ie, with software data pump). The model had 16 tasks:

- 7 modem tasks, of which 4 were kernel mode threads and 3 were user mode threads.
- 4 audio tasks, including AEC, all at user mode.

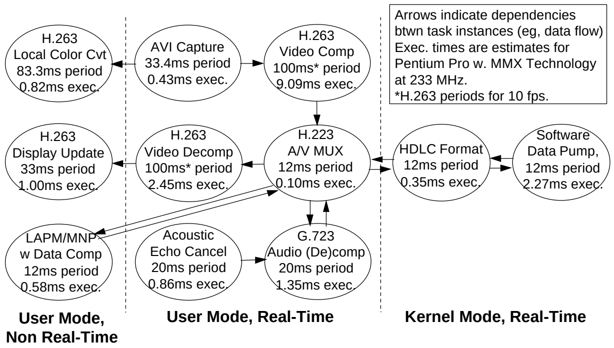


Figure 4: Task Graph for ProShare H.324 Video Teleconferencing on Pentium II Processor Based System (233 MHz)

- 5 video tasks, all at user mode.

Figure 4 shows a simplified task graph for the model where transmit and receive tasks have been coalesced into single tasks. Execution times are estimates for Pentium Pro with MMX Technology running at 233 MHz. Worst case Pentium processor utilizations for audio and video tasks were provided by the Intel Architecture Labs Algorithms group, as well as estimates of speedup due to use of MMX instructions. Linear scaling with increasing processor speed has been assumed for all application components.

The video teleconferencing model indicated that at 10 frames per second (fps) and baseline video quality the proposed application will utilize less than 50% of a 233 MHz Pentium Pro processor with MMX Technology. Under heavy load the modem should miss task deadlines infrequently, and should not suffer a loss of connection. At 200 MHz loss of connection may occur rarely under heavy load. Audio drop-outs will occur under heavy load, but no more than 1 or 2 per minute at 233 MHz.

Video jitter and frame loss should be less of a problem. Figure 5 indicates which tasks are projected to miss deadlines under heavy load for a variety of processor speeds. A good example of heavy load is launching a large application such as Microsoft Excel* or Word. Normal use of such an application does not constitute heavy load except for operations such as printing or repaginating a long document.

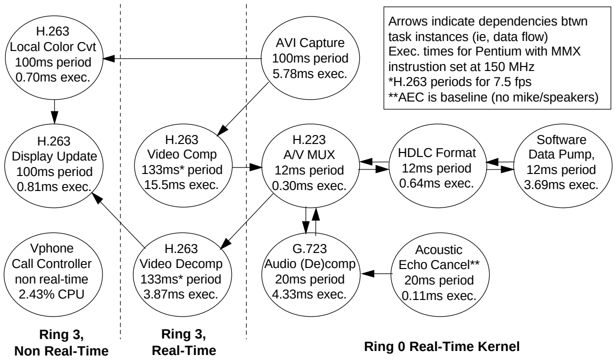


Figure 6: Task Graph for ProShare Video Teleconferencing (H.324) on Pentium Processor with MMX Technology Based System (150 MHz)

We validated the behavior predicted by our model by adapting our model to a laboratory version of a Windows 95 video teleconferencing application which uses Intel's ProShare technology and has a soft data pump modem implemented using a low latency real-time kernel which runs as a Windows 95 VxD. The task graph for this model is given in Figure 6; differences from the graph in Figure 4 are primarily due to the need to do more processing at ring 0 (in kernel mode) under Windows 95.

Processor and Speed	Pentium			Pentium Pro			
Task Name	166	200		200	233	266	300
Software Data Pump	X	?		?			
HDLC Formatting	X	X		X	?		
H.223 A/V MUX	X	X		X	X	?	
G.723 Audio Codec	X	X		X	X	X	?
Acoustic Echo Cancellation	X	X		X	X	X	X
Video Capture							
H.263 Video Decomp + CCvt							
H.263 Video Comp/Decomp							
H.263 Video Color Convert							
Display Update							

X = definite miss
? = possible miss
= not missed

Figure 5: Projected Task Deadlines Missed in the Presence of Heavy External Load for H.324 Video Teleconferencing on Pentium MMX Technology at 150 MHz

Task Name	Predicted	Observed
Software Data Pump	37.50%	~ 40%
HDLC Formatting	30.90%	~30%
H.223 A/V MUX	27.70%	30%
G.723 Audio Codec	0.00%	0%
H.263 Video Decomp + CCvt	0.25%	0%

Figure 7: Imposed External Load Necessary to Cause Missed Deadlines for H.324 Video Teleconferencing on Pentium processor with MMX Technology Based System

While a video call was in progress we progressively loaded the processor with an extremely high priority real-time task which also runs in the low latency real-time kernel and consumes a specified percentage of the processor bandwidth. This essentially simulated slowing down the processor, but with a finer granularity than is possible using the jumpers on a personal computer motherboard. On a Pentium processor with MMX Technology the model predicted the imposed high-priority load at which noticeable video jitter, loss of audio, modem retrain and modem hangup would occur within 5% of the observed load, as shown in Figure 7.

7. Conclusions

We have demonstrated a methodology for applying Rate Monotonic Analysis in a domain where it has heretofore not been applicable. It is simple and easily applied using any of a number of automated RMA tools. A variety of tools for performing schedulability analysis are commercially available and, although not designed for the type of analysis performed here, they can readily be adapted using the techniques we have described.

Static timing analysis can significantly benefit designers and developers of multimedia streaming applications. Analysis methods enable engineers to determine schedulability explicitly rather than relying on rules of thumb such as "use less than 50% of the CPU". On a non-real-time operating system such as Windows, such rules of thumb ignore the danger to schedulability posed by transient overloads under stress. Our extensions to RMA enable designers of personal computer real-time software applications to determine in the design phase, when hardware may not yet be available, if collections of applications which may severely stress the processor can nevertheless have acceptable behavior.

We designed a task set of a model of a video teleconferencing application with a fully native modem (i.e., including a software data pump) and analyzed its timing behavior using the PERTS Schedulability Analyzer. Results indicate that the modeled host-based video teleconferencing on Pentium Pro processor based systems with MMX Technology should run with adequate quality of service at 233 MHz although audio drop-outs may occur under load. We validated the behavior predicted by our model on prototype versions of a video teleconferencing application using Intel's ProShare technology with a soft data pump modem. The model accurately predicted loss of video and audio as well as modem hangup as the processor was loaded with a high-priority real-time task which consumed a specified percentage of the processor bandwidth.

8. Acknowledgments

Zdenek Brun, Lyle Cool, Derek Graham, Judi Goldstein, Atul Gupta, Roger Hurwitz, Jaya Jeyaseelan, Jeff Kidder, Sridhar Rajagopal and Kim Toll all provided assistance. K. Sridharan read an early draft of this paper and gave much useful feedback. Mary Griffith provided much needed editorial assistance and was utterly fearless in wading through the equations. Finally, the support of Tom Dingwall and the Media Infrastructure group of the Intel Architecture Labs is gratefully noted.

9. References

[1 Audsley, A., Burns, A., Richardson, M.F. and Wellings, A.J. STRESS: A Simulator for Hard Real-Time Systems, Software--Practice and Experience, 24(6), June 1994, pp. 543-564.

[2] Baker, T.P. A Stack-Based Resource Allocation Policy for Realtime Processes, Proceedings of the Real-Time Systems Symposium, IEEE, December, 1990, pp. 191-200.

[3] Business Applications Performance Corporation (BAPCo), “BAPCo Releases Application-Based Benchmark for PCs running Windows 95 and Windows NT”, Press Release by BAPCo, Santa Clara, CA, June 12, 1996.

[4] Endo, Y., Wang, Z., Chen, J.B., Seltzer, M. Using Latency to Evaluate Interactive System Performance, Proceeding of the Second Symposium on Operating Systems Design and Implementation, October 1996, USENIX Association, pp 185-199.

[5] Goodenough, J.B. and Sha, L. The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High Priority Ada Tasks. Ada Letters, 1988 Special Edition, 8(7), pp. 20-31.

[6] Harbour, M.G., Klein, M.H. and Lehoczky, J. Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority. Proceedings of the IEEE Real-Time Systems Symposium, July, 1991. IEEE, pp. 116-128.

[7] Humphrey, M., Stankovic, J. A. CAISARTS: A Tool for Real-Time Scheduling Assistance, Pro-ceedings of the Real-Time Technology and Applications Symposium, IEEE, June 1996, pp 150-9.

[8] Katcher, D.I., Arakawa, H. and Strosnider, J. Engineering and Analysis of Fixed Priority Schedulers. IEEE Transactions on Software Engineering, 19(9), September, 1993, pages 920-934.

[9] Klein, M.H. and Ralya, M.H. An Analysis of Input/Output Paradigms for Real-Time Systems. Software Engineering Institute Technical Report CMU/SEI-90-TR-19, 1990.

[10] Klein, M.H., Ralya, T., Pollak, B., Obenza, R. and Harbour, M.G. A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, Boston, MA, 1993.

[11] Liu, C.L. and Layland, J.W. Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment. JACM, 20(1), January 1973, pp. 40-61.

[12] Liu, J.W.S., Redondo, J.L., Deng, Z., Tia, T.S., Bettati, R., Silberman, A., Storch, M., Ha, R. and Shih, W.K. PERTS: A Prototyping Environment for Real-Time Systems. Proceeding of the IEEE Real-Time Systems Symposium, December 1993, IEEE, pp. 184-188.

[13] Luqi. Computer-Aided Prototyping for a Command-and-Control System using CAPS. IEEE Software, January 1992, pp. 56-67.

[14] Marko, M.A. Using Real-Time Analysis throughout the Software Development Life-Cycle. Embedded System Conference (Technical Session), September, 1995.

[15] Microsoft Corp. Real-Time Systems and Microsoft Windows NT. Microsoft Development Library White Paper, Microsoft Corp., Redmond, WA. June 1995.

[16] Ousterhout, J. K. Why Aren’t Operating Systems Getting Faster As Fast as Hardware, Proceeding of the USENIX Summer Conference, June 1990, USENIX Association, pp 247-256.

[17] Sha, L., Rajkumar, R. and Lehoczky, J.P. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, 39(9), September 1990, pp. 1175-1185.

[18] Sprunt, B., Sha, L. and Lehoczky, J. Aperiodic Task Scheduling for Hard-Real-Time Systems. J. Real-Time Systems, 1(1), pp. 27-60.

[19] Strosnider, J.K. Performance Engineering Real-Time/ Multimedia Systems, Proceeding of the Real-Time Technology and Applications Symposium (tutorial/demo) June 1996.